HoGent

BEDRIJF
EN
ORGANISATIE

Hoofdstuk 28
Netwerkprogrammeren

Leerdoelen

- Netwerkprogrammeren in Java met
  - URL's
  - Sockets
  - Datagrammen
  - Clients
  - Servers

1. Inleiding

- Fundamentele netwerkmogelijkheden:
  - package java.net
  - Stream-based communicatie: netwerking als een stream van gegevens
  - Packet-based communicatie: verzenden van individuele pakketten met informatie zoals beelden, audio en video over het Internet
- Client/server:
  - De client vraagt een actie uit te voeren
  - De server voert de actie uit
  - De server antwoordt aan de client

  → request-response model

  → vb: interactie tussen webbrowsers en webservers

- Met socket-based communicatie is netwerking zoals file I/O
  - Een socket is stukje software dat één eindpunt van een verbinding voorstelt
- Met stream sockets brengt een proces een connectie tot stand met een ander proces
- Zodra de connectie er is, stromen de gegevens tussen de processen in continue streams
- Stream sockets leveren een connectie-geörienteerde service
- Het gebruikte transmissieprotocol is TCP (Transmission Control Protocol)

- Met datagram sockets worden individuele pakketten met informatie verzonden
- UDP—User Datagram Protocol—is een connectieloze service, en zodoende wordt niet gegarandeerd dat de pakketten in een bepaalde volgorde toekomen
  - Pakketten kunnen verloren gaan of zelfs dubbel aankomen
- UDP is het meest geschikt voor netwerktoepassingen die geen error checking en betrouwbaarheid van TCP vereisen

HoGent 5

TCP versus UDP

TCP

- Transmission control protocol
- Connectie-georiënteerd
- 3-way handshaking
- Betrouwbaar transport

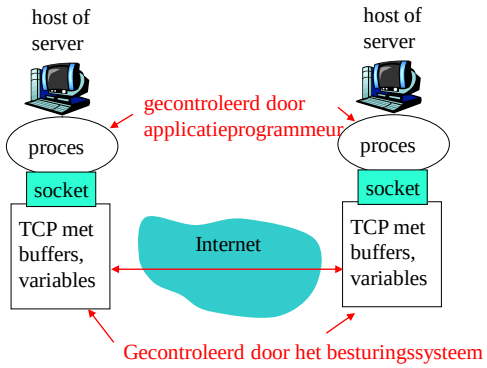
UDP

- User datagram protocol
- Connectieloos (no handshaking)
- Onbetrouwbare data tranfert
- Snelheid is belangrijk
- Realtime applicaties (skype)

HoGent 6

2. Socket-programmeren met TCP

- Schema:



HoGent

Een eenvoudige **Server** met Stream Sockets opzetten

Een eenvoudige server opzetten in 5 stappen:

Stap 1: creëer een ServerSocket object

- ServerSocket constructor
  ServerSocket server = new ServerSocket (*portNumber, queueLength);*
- De constructor legt het poortnummer vast waarop de server wacht op connecties van de client
  → binding van de server aan de poort
- Een client vraagt een connectie aan de server op deze poort
- Een geldig poortnummer ligt tussen 0 en 65565.
  De meeste besturingssystemen reserveren de poortnummers kleiner dan 1024 voor systeem services
- Een poort vragen die al in gebruik is of geen geldig nummer heeft, leidt tot een BindException

HoGent

- Programma's beheren elke connectie van een client met een Socket object.
- Sockets verbergen de complexiteit van netwerk-programmeren

***Stap 2:* de server luistert onafgebroken naar een poging van een client om een connectie te maken (blocks)**

- Het programma roept de methode accept aan om te luisteren naar een connectie van een client

```
Socket connection = server.accept();
```

  - Deze methode levert een Socket af wanneer een connectie met een client tot stand gekomen is
- Door de Socket kan de server interageren met de client

HoGent 9

Stap 3: de OutputStream- en InputStream-objecten worden opgehaald zodat de server kan communiceren met de client door het verzenden en ontvangen van bytes.

  - De server roept de methode getOutputStream aan op de Socket en krijgt een referentie naar de Socket's OutputStream. Dan wordt de methode getInputStream aangeroepen op de Socket om een referentie te krijgen naar de the Socket's InputStream

Stap 4: tijdens de verwerkingsfase communiceren de server en de client via de OutputStream- en InputStream-objecten

Stap 5: wanneer de transmissie afgehandeld is, sluit de server de connectie door de methode close aan te roepen op de streams en op de Socket

HoGent 10

Een eenvoudige **Client** met Stream Sockets opzetten

Een eenvoudige client opzetten in 4 stappen:

***Stap 1**:* de Socket constructor legt een connectie met de server

```
Socket connection = new Socket (serverAddress, port);
```

- Als de connectie tot stand gebracht is, dan wordt een Socket afgeleverd
- Als de connectie niet tot stand kan gebracht worden, dan wordt een IOException geworpen
- Een onjuiste servernaam heeft een UnknownHostException tot gevolg

- ***Stap 2:*** de client gebruikt de methoden getInputStream en getOutputStream om referenties naar InputStream and OutputStream te verkrijgen
- ***Stap 3:*** tijdens de verwerkingsfase communiceren de server en de client via de OutputStream en InputStream objecten
- ***Stap 4:*** wanneer de transmissie afgehandeld is, sluit de client de connectie door de methode close aan te roepen op de streams en op de Socket

Client/server socket interaction: TCP

Server (running on **`hostid`**)

Client

Creëer ServerSocket voor inkomende request:
ServerSocket server = new ServerSocket (p, qL)

Wacht op inkomende connectie request
Socket connection = server.accept()

TCP connection setup

creëer Socket
Socket connection = new Socket(SA, p)

Zend verzoek met connection

lees verzoek van connection

Schrijf antwoord naar connection

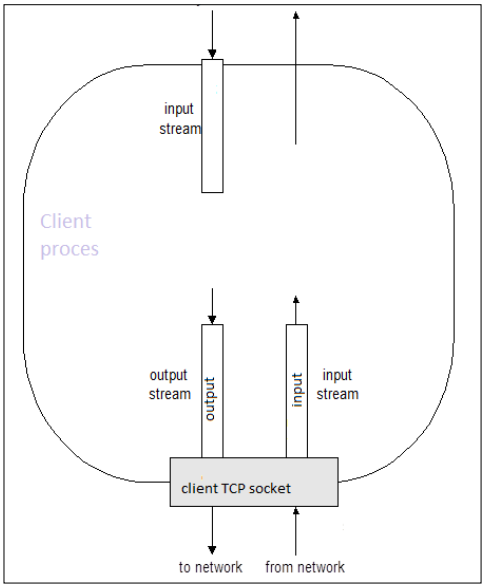
Lees antwoord van connection

sluit connection

sluit connection

HoGent

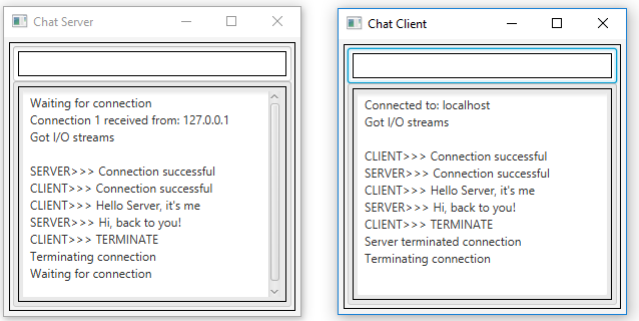
Streams bij TCP



HoGent

Client/Server Interactie met Stream Socket Connecties

- Voorbeeld: een eenvoudige client-/server chat applicatie (**zie NB projecten** OOPIII_NET_TCP_FX_server, OOPIII_NET_TCP_FX_client en OOPIII_NET_TCP_FX_common)



TCP Chat Voorbeeld

- De methode runServer zet de server klaar om een connectie te ontvangen en één connectie per keer te verwerken
- De methode getInet-Address levert een InetAddress (package java.net) af dat informatie bevat over de client
- De methode getHostName levert de hostname af van de client
- Het IP adres (127.0.0.1) en de hostname (localhost) zijn bruikbaar om netwerkapplicaties te testen op lokale computer
  - Heet het loopback adres
- De client kan uitgevoerd worden vanaf elke computer op het internet en geeft het IP adres of de hostname van de server mee als command-line argument

TCP Chat Voorbeeld

- Aanroep van de methode flush van OutputStream
  - Zorgt ervoor dat de ObjectOutputStream op de server de stream header zendt naar de ObjectInputStream van de client
  - De stream header bevat informatie van object serialization dat gebruikt wordt om objecten te verzenden
  - Deze informatie is nodig voor ObjectInputStream zodat voorbereidingen getroffen worden om die objecten correct te ontvangen
    - Creëer eerst altijd ObjectOutputStream
    - Roep dan flush aan

TCP Chat Voorbeeld

- De methode getByName levert een object af dat het IP adres bevat
  - De methode getByName heeft een String als parameter die ofwel het IP adres of de hostname bevat
- De localhost:
  - InetAddress.getByName(“127.0.0.1”)
  - InetAddress.getByName("localhost")
  - InetAddress.getLocalHost()

3. Connectieloze transmissie met datagrammen

- Connectie geörienteerde transmissie vertoont gelijkenis met het telefoonverkeer
  - Je draait een nummer en je krijgt een verbinding met de telefoon van de persoon met wie je wil communiceren
  - De verbinding blijft in stand zelfs al praat je niet
- Connectieloze transmissie met datagrammen vertoont gelijkenis met de bezorging van post via de klassieke weg (postbode)
  - Een grote boodschap die niet in één omslag past, wordt verdeeld en elk deel wordt in een genummerde briefomslag verstuurd
  - Alle omslagen worden op hetzelfde tijdstip gepost
  - De omslagen kunnen toekomen in juiste volgorde, in een andere volgorde of totaal niet toekomen (uitzonderlijk)

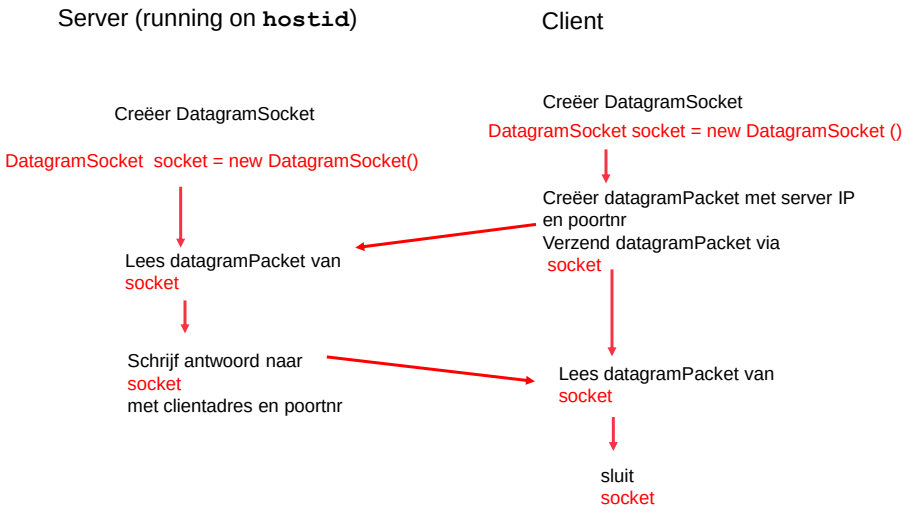
19

- De klasse Server heeft twee DatagramPakketten die de server gebruikt om informatie te verzenden en te ontvangen en één DatagramSocket die de pakketten verzendt en ontvangt
- De DatagramSocket constructor kent aan de server een poort toe waarop de server de pakketten van de client kan ontvangen
  - De client geeft het poortnummer mee in de pakketten die hij verzendt naar de server
  - Een Socket-Exception wordt gegooid wanneer de DatagramSocket constructor het DatagramSocket niet kan binden aan het poortnummer (ongeldig poortnummer, poortnummer reeds in gebruik)

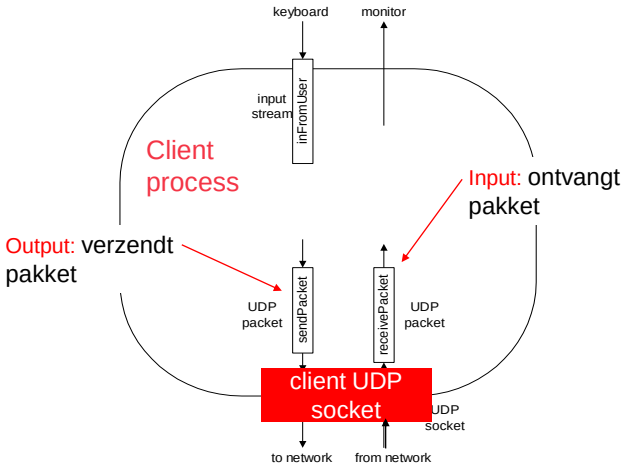
HoGent

20

Client/server socket interaction: UDP



Client bij UDP



Voorbeeld: UDP Chat

• Voorbeeld: een eenvoudige UPD versie van de client-/server chat applicatie (zie **NB projecten** OOPIII_NET_UDP_FX) De server en de client versie zitten in dezelfde applicatie. Het verschil is door een run argument mee te geven (Unnamed parameter)

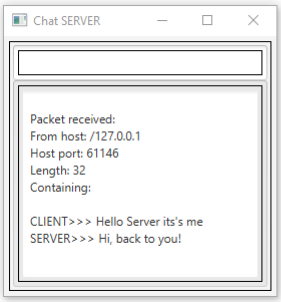
- •Als client --> arg1 = CLIENT en optioneel arg2 = hostname
- •Als server --> geen arg nodig (default status = SERVER)

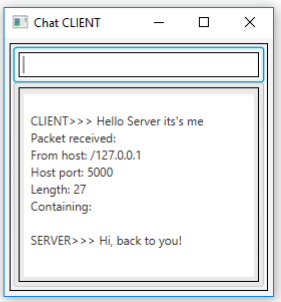
•Voor server run:

- • doe build, kopieer executable jar (zit in map dist\) naar andere map (eventueel ander computer) en run de executable jar (dubbelklik).
- • Dan kan je één of meerdere clients runnen (opgelet unnamed run parameter CLIENT instellen)

HoGent

Voorbeeld: UDP Chat





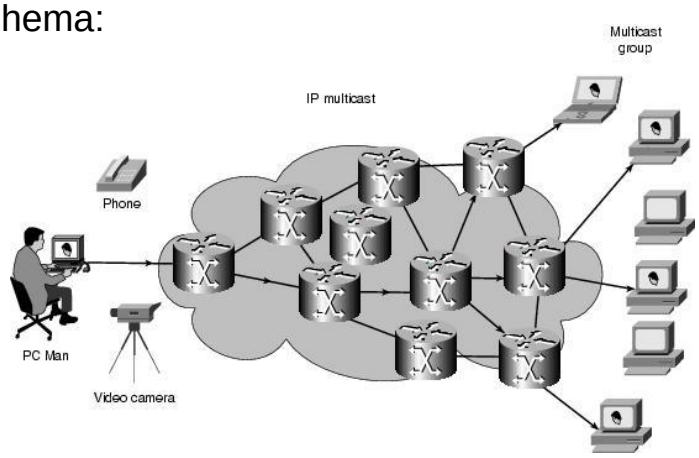
HoGent

- De methode receive wacht op een pakket dat toekomt op de Server
  - Slaat het verzonden pakket op in het DatagramPacket
  - Werpt een IOException als een error optreedt bij het ontvangen van een pakket
- De methode getAddress levert het IP adres van de computer waar het pakket werd verzonden
- De methode getPort levert het poortnummer van de client die het pakket verzond
- De methode getLength levert het aantal bytes van de ontvangen gegevens af
- De methode getData levert een byte array met de gegevens
- De methode send gooit een IOException wanneer een error plaatsvindt bij het verzenden van een pakket

25

4. Multicast

- Schema:



HoGent

26

4. Multicast

- Zendt pakketten naar een multicast-adres
- Clients kunnen aansluiten bij een multicastgroep
- Verlaten van een multicastgroep is mogelijk op elk ogenblik
- Anoniem:
  - "Server" weet niet wie er luistert
  - Iedereen kan een bericht verzenden naar een multicastgroep

UDP Multicast Server

- Stap 1: Creëer een DatagramSocket

```
DatagramSocket dgSocket = new DatagramSocket(4445);
```

- Stap 2: Kies een multicastadres

```
InetAddress group =
InetAddres.getByName("230.0.0.1");
```

- Stap 3: Maak een datagrampakket

```
DatagramPacket packet =
 new DatagramPacket(buf.getBytes(),
     buf.length, group, 4446);
```

- Stap 4: Verzend datagrampakket

```
dgSocket.send(packet);
```

- Stap 5: Sluit socket

```
dgramSocket.close();
```

UDP Multicast Server

- StartUpServer

```
public class StartupServer {

    public static void main(String[] args) throws java.io.IOException {
        new MultiCastServer().run();
    }
}
```

- MultiCastServer

UDP Multicast Server

```
public class MultiCastServer {

    protected DatagramSocket socket = null;
    protected Scanner in = null;
    protected boolean moreQuotes = true;

    public MultiCastServer() throws IOException {
        socket = new DatagramSocket(4445);
        try {
            in = new Scanner(new File("one-liners.txt"));
        } catch (FileNotFoundException e) {
            System.err.println("Could not open quote file. Serving time instead.");
        }
    }
```

UDP Multicast Server

```
private String getNextQuote() {
    String returnValue;
    if (!in.hasNextLine()) {
        in.close();
        moreQuotes = false;
        returnValue = "No more quotes. Goodbye.";
    } else {
        returnValue = in.nextLine();
    }
    return returnValue;
}
```

HoGent

31

UDP Multicast Server

```
public void run() {
    System.out.println("Server running");
    SecureRandom random = new SecureRandom();
    String dString;
    while (moreQuotes) {
        try {
            if (in == null) {
                dString = new Date().toString();
            } else {
                dString = getNextQuote();
            }
            byte[] buf = dString.getBytes();
```

HoGent

32

UDP Multicast Server

```
            InetAddress group = InetAddress.getByName("230.0.0.1");
            DatagramPacket packet = new DatagramPacket(buf, buf.length, group, 4446);
            socket.send(packet);
            try {
                Thread.sleep(random.nextInt(3000) + 2000);
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
        } catch (IOException e) {
            e.printStackTrace();
            moreQuotes = false;
        }
    }
    socket.close();
}
```

HoGent

UDP Multicast Client

- Stap 1: Maak een `MulticastSocket` object

```
MulticastSocket socket = new MulticastSocket(4446);
```

- Stap 2: Sluit aan bij een multicastgroep

```
InetAddress addr = InetAddres.getByName("230.0.0.1");
socket.joinGroup(addr);
```

- Stap 3: Ontvang een datagrambericht

```
DatagramPacket packet =
  new DatagramPacket(buf, buf.length);
socket.receive(packet);
```

- Stap 4: Verlaat de multicastgroep

```
socket.leaveGroup(address);
```

- Stap 5: Sluit socket

```
socket.close();
```

UDP Multicast Client

```
public class StartUpClient {

    public static void main(String[] args) throws IOException {
        new MultiCastClient().run();
    }

}
```


UDP Multicast Client

```
public class MultiCastClient {
    public void run() throws IOException {
        MulticastSocket socket = new MulticastSocket(4446);
        InetAddress address = InetAddress.getByName("230.0.0.1");
        DatagramPacket packet;
        byte[] buf = new byte[256];
        socket.joinGroup(address);
        System.out.println("Client running");
        for (int i = 0; i < 5; i++) {
            packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            String received = new String(packet.getData(), 0, packet.getLength());
            System.out.println("Quote of the moment: " + received);
        }
        socket.leaveGroup(address);
        socket.close();
    }
}
```

UDP Multicast Demo

- Start Server
- Start Client

```
Quote of the Moment: Give me ambiguity or give me something else.
Quote of the Moment: I.R.S.: We've got what it takes to take what you've got!
Quote of the Moment: We are born naked, wet and hungry. Then things get worse.
Quote of the Moment: Make it idiot proof and someone will make a better idiot.
Quote of the Moment: He who laughs last thinks slowest!
```

HoGent

37

5. Client/Server Tic-Tac-Toe met een Multithreaded Server

- Voorbeeld uit het boek dat met Swing werkt voor de gui.

  De gui is hier niet van belang. **De focus** ligt hier wel op het multithreadingdeel in combinatie met netwerkprogrammeren. Het zijn deze aspecten die hier toegelicht worden. Er wordt gebruik gemaakt van conditionobjecten om het spel te starten (er moeten twee spelers zijn) en om de spelers te synchroniseren (ieder om beurt).

- Tic-Tac-Toe geïmplementeerd door gebruik van client/server technieken met stream sockets.

- De TicTacToeServer ontvangt van elke client een connection. Er wordt een instantie van Player gemaakt om de client in een afzonderlijke thread te verwerken.

38

5. Client/Server Tic-Tac-Toe met een Multithreaded Server

- Deze threads laten de clients onafhankelijk het spel spelen.
- De eerste client die geconnecteerd is met de server is speler X en de tweede is speler O.
- Speler X is eerst aan de beurt.
- De server houdt de gegevens bij over het speelbord en controleert of de spelers geldige zetten maken.

39

Voorbeeld

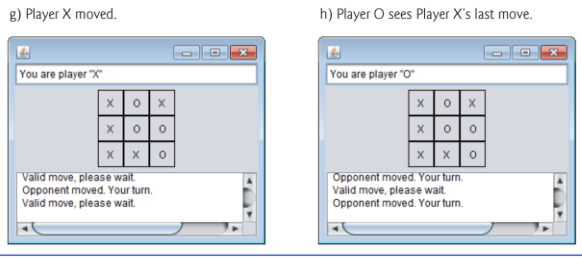


Fig. 27.17 | Sample outputs from the client/server Tic-Tac-Toe program. (Part 4 of 4.)

HoGent

40

```
1  // Fig. 27.13: TicTacToeServer.java
2  // Server side of client/server Tic-Tac-Toe program.
3  import java.awt.BorderLayout;
4  import java.net.ServerSocket;
5  import java.net.Socket;
6  import java.io.IOException;
7  import java.util.Formatter;
8  import java.util.Scanner;
9  import java.util.concurrent.ExecutorService;
10 import java.util.concurrent.Executors;
11 import java.util.concurrent.locks.Lock;
12 import java.util.concurrent.locks.ReentrantLock;
13 import java.util.concurrent.locks.Condition;
14 import javax.swing.JFrame;
15 import javax.swing.JTextArea;
16 import javax.swing.SwingUtilities;
17
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 1 of 17.)

HoGent

41

```
18 public class TicTacToeServer extends JFrame
19 {
20    private String[] board = new String[ 9 ]; // tic-tac-toe board
21    private JTextArea outputArea; // for outputting moves
22    private Player[] players; // array of Players
23    private ServerSocket server; // server socket to connect with clients
24    private int currentPlayer; // keeps track of player with current move
25    private final static int PLAYER_X = 0; // constant for first player
26    private final static int PLAYER_O = 1; // constant for second player
27    private final static String[] MARKS = { "X", "O" }; // array of marks
28    private ExecutorService runGame; // will run players
29    private Lock gameLock; // to lock game for synchronization
30    private Condition otherPlayerConnected; // to wait for other player
31    private Condition otherPlayerTurn; // to wait for other player's turn
32
33    // set up tic-tac-toe server and GUI that displays messages
34    public TicTacToeServer()
35    {
36       super( "Tic-Tac-Toe Server" ); // set title of window
37
38       // create ExecutorService with a thread for each player
39       runGame = Executors.newFixedThreadPool( 2 );
40       gameLock = new ReentrantLock(); // create lock for game
41
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 2 of 17.)

HoGent

42

```
42        // condition variable for both players being connected
43        otherPlayerConnected = gameLock.newCondition();
44
45        // condition variable for the other player's turn
46        otherPlayerTurn = gameLock.newCondition();
47
48        for ( int i = 0; i < 9; i++ )
49           board[ i ] = new String( "" ); // create tic-tac-toe board
50        players = new Player[ 2 ]; // create array of players
51        currentPlayer = PLAYER_X; // set current player to first player
52
53        try
54        {
55           server = new ServerSocket( 12345, 2 ); // set up ServerSocket
56        } // end try
57        catch ( IOException ioException )
58        {
59           ioException.printStackTrace();
60           System.exit( 1 );
61        } // end catch
62
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 3 of 17.)


```
63        outputArea = new JTextArea(); // create JTextArea for output
64        add( outputArea, BorderLayout.CENTER );
65        outputArea.setText( "Server awaiting connections\n" );
66
67        setSize( 300, 300 ); // set size of window
68        setVisible( true ); // show window
69     } // end TicTacToeServer constructor
70
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 4 of 17.)

```
71     // wait for two connections so game can be played
72     public void execute()
73     {
74        // wait for each client to connect
75        for ( int i = 0; i < players.length; i++ )
76        {
77           try // wait for connection, create Player, start runnable
78           {
79              players[ i ] = new Player( server.accept(), i );
80              runGame.execute( players[ i ] ); // execute player runnable
81           } // end try
82           catch ( IOException ioException )
83           {
84              ioException.printStackTrace();
85              System.exit( 1 );
86           } // end catch
87        } // end for
88
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 5 of 17.)

HoGent

45

```
89        gameLock.lock(); // lock game to signal player X's thread
90
91        try
92        {
93           players[ PLAYER_X ].setSuspended( false ); // resume player X
94           otherPlayerConnected.signal(); // wake up player X's thread
95        } // end try
96        finally
97        {
98           gameLock.unlock(); // unlock game after signalling player X
99        } // end finally
100    } // end method execute
101
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 6 of 17.)

HoGent

46

```
102    // display message in outputArea
103    private void displayMessage( final String messageToDisplay )
104    {
105       // display message from event-dispatch thread of execution
106       SwingUtilities.invokeLater(
107          new Runnable()
108          {
109             public void run() // updates outputArea
110             {
111                outputArea.append( messageToDisplay ); // add message
112             } // end  method run
113          } // end inner class
114       ); // end call to SwingUtilities.invokeLater
115    } // end method displayMessage
116
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 7 of 17.)

HoGent

47

```
117    // determine if move is valid
118    public boolean validateAndMove( int location, int player )
119    {
120       // while not current player, must wait for turn
121       while ( player != currentPlayer )
122       {
123          gameLock.lock(); // lock game to wait for other player to go
124
125          try
126          {
127             otherPlayerTurn.await(); // wait for player's turn
128          } // end try
129          catch ( InterruptedException exception )
130          {
131             exception.printStackTrace();
132          } // end catch
133          finally
134          {
135             gameLock.unlock(); // unlock game after waiting
136          } // end finally
137       } // end while
138
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 8 of 17.)

HoGent

48

```
139        // if location not occupied, make move
140        if ( !isOccupied( location ) )
141        {
142           board[ location ] = MARKS[ currentPlayer ]; // set move on board
143           currentPlayer = ( currentPlayer + 1 ) % 2; // change player
144
145           // let new current player know that move occurred
146           players[ currentPlayer ].otherPlayerMoved( location );
147
148           gameLock.lock(); // lock game to signal other player to go
149
150           try
151           {
152              otherPlayerTurn.signal(); // signal other player to continue
153           } // end try
154           finally
155           {
156              gameLock.unlock(); // unlock game after signaling
157           } // end finally
158
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 9 of 17.)

HoGent

49

```
159           return true; // notify player that move was valid
160        } // end if
161        else // move was not valid
162           return false; // notify player that move was invalid
163     } // end method validateAndMove
164
165     // determine whether location is occupied
166     public boolean isOccupied( int location )
167     {
168        if ( board[ location ].equals( MARKS[ PLAYER_X ] ) ||
169           board [ location ].equals( MARKS[ PLAYER_O ] ) )
170           return true; // location is occupied
171        else
172           return false; // location is not occupied
173     } // end method isOccupied
174
175     // place code in this method to determine whether game over
176     public boolean isGameOver()
177     {
178        return false; // this is left as an exercise
179     } // end method isGameOver
180
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 10 of 17.)

HoGent

50

```
181    // private inner class Player manages each Player as a runnable
182    private class Player implements Runnable
183    {
184       private Socket connection; // connection to client
185       private Scanner input; // input from client
186       private Formatter output; // output to client
187       private int playerNumber; // tracks which player this is
188       private String mark; // mark for this player
189       private boolean suspended = true; // whether thread is suspended
190
191       // set up Player thread
192       public Player( Socket socket, int number )
193       {
194          playerNumber = number; // store this player's number
195          mark = MARKS[ playerNumber ]; // specify player's mark
196          connection = socket; // store socket for client
197
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 11 of 17.)

HoGent


```
198          try // obtain streams from Socket
199          {
200             input = new Scanner( connection.getInputStream() );
201             output = new Formatter( connection.getOutputStream() );
202          } // end try
203          catch ( IOException ioException )
204          {
205             ioException.printStackTrace();
206             System.exit( 1 );
207          } // end catch
208       } // end Player constructor
209
210       // send message that other player moved
211       public void otherPlayerMoved( int location )
212       {
213          output.format( "Opponent moved\n" );
214          output.format( "%d\n", location ); // send location of move
215          output.flush(); // flush output
216       } // end method otherPlayerMoved
217
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 12 of 17.)

HoGent

```
218       // control thread's execution
219       public void run()
220       {
221          // send client its mark (X or O), process messages from client
222          try
223          {
224             displayMessage( "Player " + mark + " connected\n" );
225             output.format( "%s\n", mark ); // send player's mark
226             output.flush(); // flush output
227
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 13 of 17.)

HoGent


```
228             // if player X, wait for another player to arrive
229             if ( playerNumber == PLAYER_X )
230             {
231                output.format( "%s\n%s", "Player X connected",
232                   "Waiting for another player\n" );
233                output.flush(); // flush output
234
235                gameLock.lock(); // lock game to  wait for second player
236
237                try
238                {
239                   while( suspended )
240                   {
241                      otherPlayerConnected.await(); // wait for player O
242                   } // end while
243                } // end try
244                catch ( InterruptedException exception )
245                {
246                   exception.printStackTrace();
247                } // end catch
248                finally
249                {
250                   gameLock.unlock(); // unlock game after second player
251                } // end finally
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 14 of 17.)

HoGent

```
252
253                // send message that other player connected
254                output.format( "Other player connected. Your move.\n" );
255                output.flush(); // flush output
256             } // end if
257             else
258             {
259                output.format( "Player O connected, please wait\n" );
260                output.flush(); // flush output
261             } // end else
262
263             // while game not over
264             while ( !isGameOver() )
265             {
266                int location = 0; // initialize move location
267
268                if ( input.hasNext() )
269                   location = input.nextInt(); // get move location
270
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 15 of 17.)

HoGent

55

```
271                // check for valid move
272                if ( validateAndMove( location, playerNumber ) )
273                {
274                   displayMessage( "\nlocation: " + location );
275                   output.format( "Valid move.\n" ); // notify client
276                   output.flush(); // flush output
277                } // end if
278                else // move was invalid
279                {
280                   output.format( "Invalid move, try again\n" );
281                   output.flush(); // flush output
282                } // end else
283             } // end while
284          } // end try
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 16 of 17.)

HoGent

56

```
285          finally
286          {
287             try
288             {
289                connection.close(); // close connection to client
290             } // end try
291             catch ( IOException ioException )
292             {
293                ioException.printStackTrace();
294                System.exit( 1 );
295             } // end catch
296          } // end finally
297       } // end method run
298
299       // set whether or not thread is suspended
300       public void setSuspended( boolean status )
301       {
302          suspended = status; // set value of suspended
303       } // end method setSuspended
304    } // end class Player
305 } // end class TicTacToeServer
```

Fig. 27.13 | Server side of client/server Tic-Tac-Toe program. (Part 17 of 17.)

HoGent


```
1  // Fig. 27.14: TicTacToeServerTest.java
2  // Class that tests Tic-Tac-Toe server.
3  import javax.swing.JFrame;
4
5  public class TicTacToeServerTest
6  {
7     public static void main( String[] args )
8     {
9        TicTacToeServer application = new TicTacToeServer();
10       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
11       application.execute();
12    } // end main
13 } // end class TicTacToeServerTest
```

Fig. 27.14 | Class that tests Tic-Tac-Toe server. (Part 1 of 2.)

HoGent

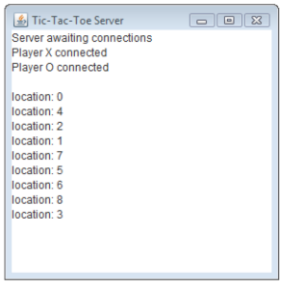


Fig. 27.14 | Class that tests Tic-Tac-Toe server. (Part 2 of 2.)

HoGent

- Wanneer een client connecteert, wordt een nieuw Player object gecreëerd om de connectie in een afzonderlijke thread te beheren en de thread wordt uitgevoerd in de runGame thread pool.
- De constructor van Player ontvangt het Socket object dat de connectie met de client weergeeft en haalt de geassocieerde input- en output- streams op.
- De methode run van Player controleert de informatie die verzonden en ontvangen wordt van de client.

HoGent

```
1  // Fig. 27.15: TicTacToeClient.java
2  // Client side of client/server Tic-Tac-Toe program.
3  import java.awt.BorderLayout;
4  import java.awt.Dimension;
5  import java.awt.Graphics;
6  import java.awt.GridLayout;
7  import java.awt.event.MouseAdapter;
8  import java.awt.event.MouseEvent;
9  import java.net.Socket;
10 import java.net.InetAddress;
11 import java.io.IOException;
12 import javax.swing.JFrame;
13 import javax.swing.JPanel;
14 import javax.swing.JScrollPane;
15 import javax.swing.JTextArea;
16 import javax.swing.JTextField;
17 import javax.swing.SwingUtilities;
18 import java.util.Formatter;
19 import java.util.Scanner;
20 import java.util.concurrent.Executors;
21 import java.util.concurrent.ExecutorService;
22
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 1 of 14.)

HoGent


```
23 public class TicTacToeClient extends JFrame implements Runnable
24 {
25    private JTextField idField; // textfield to display player's mark
26    private JTextArea displayArea; // JTextArea to display output
27    private JPanel boardPanel; // panel for tic-tac-toe board
28    private JPanel panel2; // panel to hold board
29    private Square[][] board; // tic-tac-toe board
30    private Square currentSquare; // current square
31    private Socket connection; // connection to server
32    private Scanner input; // input from server
33    private Formatter output; // output to server
34    private String ticTacToeHost; // host name for server
35    private String myMark; // this client's mark
36    private boolean myTurn; // determines which client's turn it is
37    private final String X_MARK = "X"; // mark for first client
38    private final String O_MARK = "O"; // mark for second client
39
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 2 of 14.)

HoGent

```
40    // set up user-interface and board
41    public TicTacToeClient( String host )
42    {
43       ticTacToeHost = host; // set name of server
44       displayArea = new JTextArea( 4, 30 ); // set up JTextArea
45       displayArea.setEditable( false );
46       add( new JScrollPane( displayArea ), BorderLayout.SOUTH );
47
48       boardPanel = new JPanel(); // set up panel for squares in board
49       boardPanel.setLayout( new GridLayout( 3, 3, 0, 0 ) );
50
51       board = new Square[ 3 ][ 3 ]; // create board
52
53       // loop over the rows in the board
54       for ( int row = 0; row < board.length; row++ )
55       {
56          // loop over the columns in the board
57          for ( int column = 0; column < board[ row ].length; column++ )
58          {
59             // create square
60             board[ row ][ column ] = new Square( ' ', row * 3 + column );
61             boardPanel.add( board[ row ][ column ] ); // add square
62          } // end inner for
63       } // end outer for
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 3 of 14.)

HoGent

63

```
64
65       idField = new JTextField(); // set up textfield
66       idField.setEditable( false );
67       add( idField, BorderLayout.NORTH );
68
69       panel2 = new JPanel(); // set up panel to contain boardPanel
70       panel2.add( boardPanel, BorderLayout.CENTER ); // add board panel
71       add( panel2, BorderLayout.CENTER ); // add container panel
72
73       setSize( 300, 225 ); // set size of window
74       setVisible( true ); // show window
75
76       startClient();
77    } // end TicTacToeClient constructor
78
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 4 of 14.)

HoGent

64

```
79     // start the client thread
80     public void startClient()
81     {
82        try // connect to server and get streams
83        {
84           // make connection to server
85           connection = new Socket(
86              InetAddress.getByName( ticTacToeHost ), 12345 );
87
88           // get streams for input and output
89           input = new Scanner( connection.getInputStream() );
90           output = new Formatter( connection.getOutputStream() );
91        } // end try
92        catch ( IOException ioException )
93        {
94           ioException.printStackTrace();
95        } // end catch
96
97        // create and start worker thread for this client
98        ExecutorService worker = Executors.newFixedThreadPool( 1 );
99        worker.execute( this ); // execute client
100    } // end method startClient
101
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 5 of 14.)

HoGent

65

```
102    // control thread that allows continuous update of displayArea
103    public void run()
104    {
105       myMark = input.nextLine(); // get player's mark (X or O)
106
107       SwingUtilities.invokeLater(
108          new Runnable()
109          {
110             public void run()
111             {
112                // display player's mark
113                idField.setText( "You are player \"" + myMark + "\"" );
114             } // end method run
115          } // end anonymous inner class
116       ); // end call to SwingUtilities.invokeLater
117
118       myTurn = ( myMark.equals( X_MARK ) ); // determine if client's turn
119
120       // receive messages sent to client and output them
121       while ( true )
122       {
123          if ( input.hasNextLine() )
124             processMessage( input.nextLine() );
125       } // end while
126    } // end method run
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 6 of 14.)

HoGent

66

```
127
128    // process messages received by client
129    private void processMessage( String message )
130    {
131       // valid move occurred
132       if ( message.equals( "Valid move." ) )
133       {
134          displayMessage( "Valid move, please wait.\n" );
135          setMark( currentSquare, myMark ); // set mark in square
136       } // end if
137       else if ( message.equals( "Invalid move, try again" ) )
138       {
139          displayMessage( message + "\n" ); // display invalid move
140          myTurn = true; // still this client's turn
141       } // end else if
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 7 of 14.)

HoGent


```
142       else if ( message.equals( "Opponent moved" ) )
143       {
144          int location = input.nextInt(); // get move location
145          input.nextLine(); // skip newline after int location
146          int row = location / 3; // calculate row
147          int column = location % 3; // calculate column
148
149          setMark(  board[ row ][ column ],
150             ( myMark.equals( X_MARK ) ? O_MARK : X_MARK ) ); // mark move
151          displayMessage( "Opponent moved. Your turn.\n" );
152          myTurn = true; // now this client's turn
153       } // end else if
154       else
155          displayMessage( message + "\n" ); // display the message
156    } // end method processMessage
157
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 8 of 14.)

HoGent

```
158    // manipulate displayArea in event-dispatch thread
159    private void displayMessage( final String messageToDisplay )
160    {
161       SwingUtilities.invokeLater(
162          new Runnable()
163          {
164             public void run()
165             {
166                displayArea.append( messageToDisplay ); // updates output
167             } // end method run
168          }  // end inner class
169       ); // end call to SwingUtilities.invokeLater
170    } // end method displayMessage
171
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 9 of 14.)

HoGent

69

```
172    // utility method to set mark on board in event-dispatch thread
173    private void setMark( final Square squareToMark, final String mark )
174    {
175       SwingUtilities.invokeLater(
176          new Runnable()
177          {
178             public void run()
179             {
180                squareToMark.setMark( mark ); // set mark in square
181             } // end method run
182          } // end anonymous inner class
183       ); // end call to SwingUtilities.invokeLater
184    } // end method setMark
185
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 10 of 14.)

HoGent

70

```
186    // send message to server indicating clicked square
187    public void sendClickedSquare( int location )
188    {
189       // if it is my turn
190       if ( myTurn )
191       {
192          output.format( "%d\n", location ); // send location to server
193          output.flush();
194          myTurn = false; // not my turn any more
195       } // end if
196    } // end method sendClickedSquare
197
198    // set current Square
199    public void setCurrentSquare( Square square )
200    {
201       currentSquare = square; // set current square to argument
202    } // end method setCurrentSquare
203
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 11 of 14.)

HoGent

71

```
204    // private inner class for the squares on the board
205    private class Square extends JPanel
206    {
207       private String mark; // mark to be drawn in this square
208       private int location; // location of square
209
210       public Square( String squareMark, int squareLocation )
211       {
212          mark = squareMark; // set mark for this square
213          location = squareLocation; // set location of this square
214
215          addMouseListener(
216             new MouseAdapter()
217             {
218                public void mouseReleased( MouseEvent e )
219                {
220                   setCurrentSquare( Square.this ); // set current square
221
222                   // send location of this square
223                   sendClickedSquare( getSquareLocation() );
224                } // end method mouseReleased
225             } // end anonymous inner class
226          ); // end call to addMouseListener
227       } // end Square constructor
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 12 of 14.)

HoGent

72

```
228
229        // return preferred size of Square
230        public Dimension getPreferredSize()
231        {
232           return new Dimension( 30, 30 ); // return preferred size
233        } // end method getPreferredSize
234
235        // return minimum size of Square
236        public Dimension getMinimumSize()
237        {
238           return getPreferredSize(); // return preferred size
239        } // end method getMinimumSize
240
241        // set mark for Square
242        public void setMark( String newMark )
243        {
244           mark = newMark; // set mark of square
245           repaint(); // repaint square
246        } // end method setMark
247
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 13 of 14.)

HoGent

73

```
248        // return Square location
249        public int getSquareLocation()
250        {
251           return location; // return location of square
252        } // end method getSquareLocation
253
254        // draw Square
255        public void paintComponent( Graphics g )
256        {
257           super.paintComponent( g );
258
259           g.drawRect( 0, 0, 29, 29 ); // draw square
260           g.drawString( mark, 11, 20 ); // draw mark
261        } // end method paintComponent
262     } // end inner-class Square
263  } // end class TicTacToeClient
```

Fig. 27.15 | Client side of client/server Tic-Tac-Toe program. (Part 14 of 14.)

HoGent

74

```
1  // Fig. 27.16: TicTacToeClientTest.java
2  // Test class for Tic-Tac-Toe client.
3  import javax.swing.JFrame;
4
5  public class TicTacToeClientTest
6  {
7     public static void main( String[] args )
8     {
9        TicTacToeClient application; // declare client application
10
11       // if no command line args
12       if ( args.length == 0 )
13          application = new TicTacToeClient( "127.0.0.1" ); // localhost
14       else
15          application = new TicTacToeClient( args[ 0 ] ); // use args
16
17       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
18    } // end main
19 } // end class TicTacToeClientTest
```

Fig. 27.16 | Test class for Tic-Tac-Toe client.

HoGent

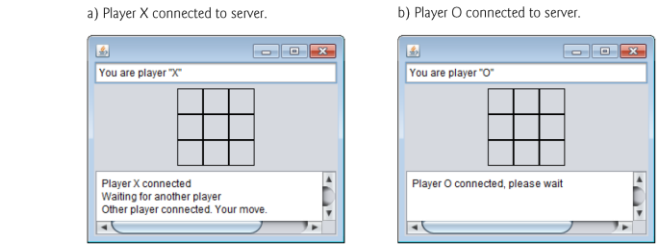
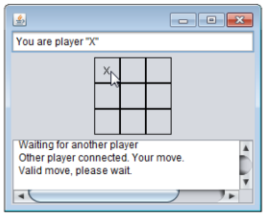


Fig. 27.17 | Sample outputs from the client/server Tic-Tac-Toe program. (Part 1 of 4.)

HoGent

c) Player X moved.



d) Player O sees Player X's move.

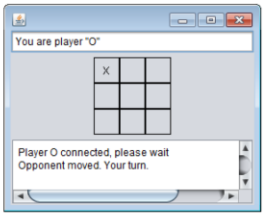
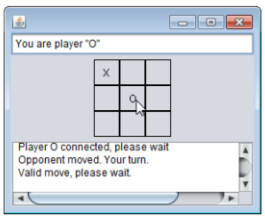


Fig. 27.17 | Sample outputs from the client/server Tic-Tac-Toe program. (Part 2 of 4.)

HoGent

77

e) Player O moved.



f) Player X sees Player O's move.

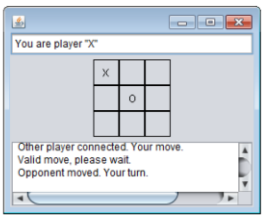


Fig. 27.17 | Sample outputs from the client/server Tic-Tac-Toe program. (Part 3 of 4.)

HoGent

78